МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное
образовательное учреждение высшего образования
«Самарский национальный исследовательский университет
имени академика С.П. Королева»
(Самарский университет)

ОТЧЕТ ПО
ЛАБОРАТОРНОЙ РАБОТЕ № 2

«Разработка набор классов для работы с функциями одной переменной, заданными в табличной форме»

по курсу
Объектно-ориентированное программирование

Выполнила: Яньшина Анастасия Юрьевна
Группа 6203-010302D

Содержание

Задание 1

Для начала создаём пакет `functions`. В нём в дальнейшем будут размещаться нужные нам классы. (Рис. 1).

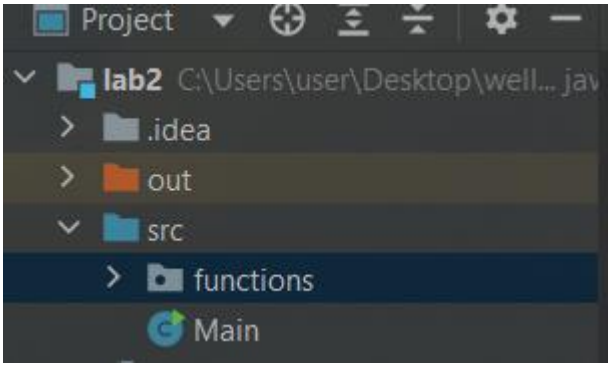


Рис. 1

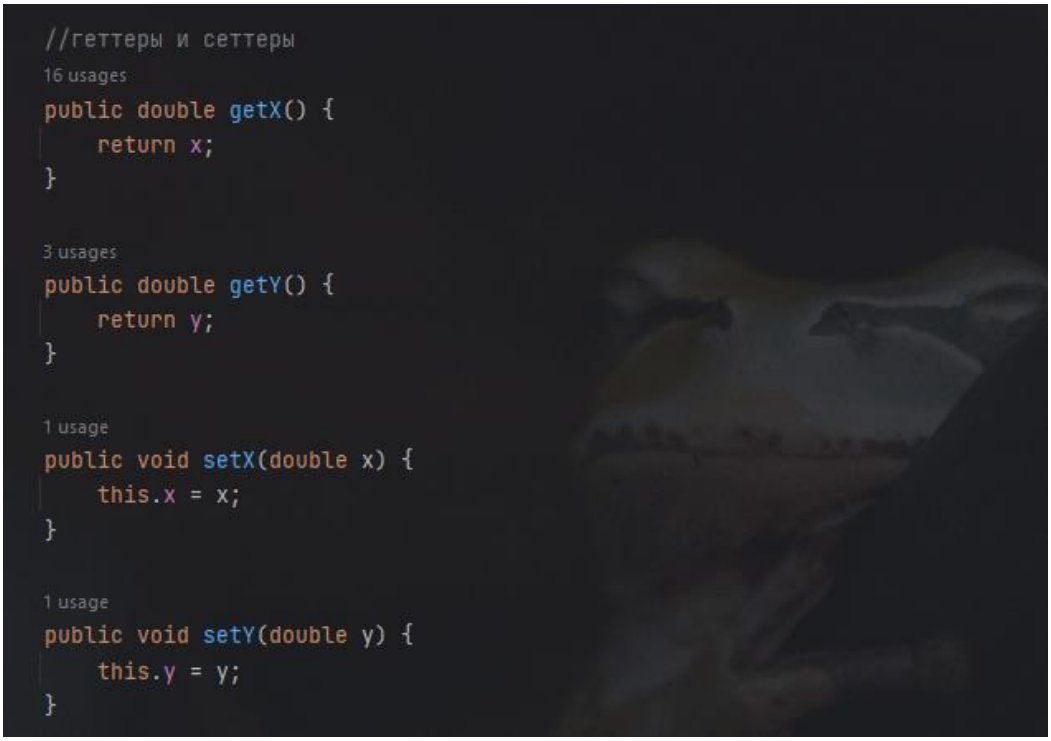
Задание 2

В пакете `functions` создаём класс `FunctionPoint`, объект которого будет описывать одну точку табулированной функции. Координаты точки по осям абсцисс и ординат реализуем в виде полей класса. (Рис. 2)

Учитывая особенности инкапсуляции, делаем поля приватными, а доступ к ним реализуем через геттеры и сеттеры (Рис. 3). Также реализуем три типа конструкторов: конструктор по умолчанию с координатами точки (0.0, 0.0), конструктор копирования и конструктор для точки с указанными координатами (Также рис. 2).

```
package functions;

23 usages
public class FunctionPoint {

    //координаты точки
    6 usages
    private double x;
    6 usages
    private double y;

    //создаёт объект точки с заданными координатами
    6 usages
    public FunctionPoint(double x, double y){
        this.x=x;
        this.y=y;
    }

    //создаёт объект точки с теми же координатами, что у указанной точки
    3 usages
    public FunctionPoint(FunctionPoint point){
        this.x=point.x;
        this.y=point.y;
    }

    //создаёт точку с координатами (0; 0)
    no usages
    public FunctionPoint() {
        this.x=0.0;
        this.y=0.0;
    }
```

Рис. 2



```
//геттеры и сеттеры
16 usages
public double getX() {
    return x;
}

3 usages
public double getY() {
    return y;
}

1 usage
public void setX(double x) {
    this.x = x;
}

1 usage
public void setY(double y) {
    this.y = y;
}
```

Рис. 3

Задание 3

Всё в том же пакете создаём класс `TabulatedFunction`, который будет описывать табулированную функцию. В качестве приватных значений класс будет хранить массив точек, принадлежащих данной табулированной функции. точки в нём упорядочены по значению координаты `x` (Рис. 4).

По заданию реализуем два конструктора: для создания функции по левой и правой границам области определения, а также количеству точек; для создания функции по левой и правой границам области определения и массиву значений (Рис. 5).

В обоих случаях для создания точек используем равные интервалы по `x`. В случае, если точка всего одна, определяем её по середине между двумя границами.

```
package functions;

4 usages
public class TabulatedFunction {

    44 usages
    private FunctionPoint[] points;
```

Рис. 4

```java
//создаёт объект табулированной функции по заданным левой и правой границе области определения, а также количеству точек для табулирования
1 usage
public TabulatedFunction(double leftX, double rightX, int pointsCount) {
    this.points = new FunctionPoint[pointsCount];
    if (pointsCount == 1) points[0] = new FunctionPoint( x: (rightX-leftX)/2,  y: 0);
    else {
        //вычисляем шаг между точками и заполняем массив точек
        double step = (rightX - leftX) / (pointsCount - 1);
        for (int i = 0; i < pointsCount; i++) {
            points[i] = new FunctionPoint( x: leftX + i * step,  y: 0);
        }
    }
}

//создаёт объект табулированной функции по заданным левой и правой границе области определения, а также значениям функции
no usages
public TabulatedFunction(double leftX, double rightX, double[] values){
    this.points = new FunctionPoint[values.length];
    if (values.length == 1) points[0] = new FunctionPoint( x: (rightX-leftX)/2, values[0]);
    else {
        //вычисляем шаг между точками и заполняем массив точек
        double step = (rightX - leftX) / (values.length - 1);
        for (int i = 0; i < values.length; i++) {
            points[i] = new FunctionPoint( x: leftX + i * step, values[i]);
        }
    }
}
```

Рис. 5

Задание 4

По заданию лабораторной добавляем нужные для работы с функцией публичные методы: `getLeftDomainBorder()`, `getRightDomainBorder()`, `getFunctionValue(double x)`. Первые два метода возвращают значения границ области определения, левой и правой соответственно (Рис. 6).

Третий метод возвращает значение функции в точке `x`. При расчёте значения пользуемся линейной интерполяцией, как и указано в задании, а также уравнением прямой, проходящей через две заданные точки $(y - y_1)/(y_2 - y_1) = (x - x_1)/(x_2 - x_1)$ (Рис. 7).

В случае, если лежит вне области определения функции, возвращаем значение `NaN` (`Not-a-Number`).

```
//возвращает левую границу области определения функции
2 usages
public double getLeftDomainBorder(){
    return points[0].getX();
}

//возвращает правую границу области определения функции
2 usages
public double getRightDomainBorder(){
    return points[points.length-1].getX();
}
```

Рис. 6

```java
//вычисляет значение функции в заданной точке x с помощью линейной интерполяции
//если x вне области определения, то возвращаем Double.NaN (Not-a-Number)
1 usage
public double getFunctionValue(double x) {
    if (x < getLeftDomainBorder() || x > getRightDomainBorder()) return Double.NaN;

    for (int i = 0; i< points.length-1; i++) {
        if (points[i].getX() <= x && x <= points[i + 1].getX()) {
            double x1 = points[i].getX();
            double x2 = points[i + 1].getX();
            double y1 = points[i].getY();
            double y2 = points[i + 1].getY();
            return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
        }
    }
    return Double.NaN;
}
```

Рис. 7

Задание 5

Также добавляем публичные методы, необходимые для работы с конкретными точками табулированной функции.

Метод `getPointsCount()` возвращает количество точек функции, метод `getPoint(int index)` возвращает копию точки по указанному индексу (Рис. 8). Отдельно для конкретных координат `getPointX (int index)` возвращающий значение абсциссы точки с указанным номером и, аналогично, `getPointY(int index)` для возвращающий значение ординаты точки с указанным номером (Рис. 9).

Помимо методов, возвращающих значения, наш класс будет иметь методы для изменения точек: `setPoint(int index, FunctionPoint point)` для замены указанной точки табулированной функции на переданную, а также `setPointX(int index, double x)` и `setPointY(int index, double y)` для замены конкретных координат (`x` или `y` соответственно) в указанной точке (Рис. 9).

Важно отметить, что методы `setPoint(int index, FunctionPoint point)` и `setPointX(int index, double x)` не изменяют значения точки, если переданный `x` лежит вне интервала, определяемого значениями соседних точек табулированной функции.

```
//возвращает количество точек табулирования
3 usages
public int getPointsCount(){return points.length;}

//возвращает копию точки по указанному индексу
no usages
public FunctionPoint getPoint(int index){return new FunctionPoint(points[index]);}
```

Рис. 8

```
//устанавливает точку по указанному индексу (с проверкой порядка по X)
1 usage
public void setPoint(int index, FunctionPoint point){
    double newX = point.getX();
    if (index > 0 && newX<=points[index-1].getX()) return;
    if (index < points.length-1 && newX >= points[index + 1].getX()) return;
    points[index] = new FunctionPoint(point);
}

//возвращает координату X точки по указанному индексу
4 usages
public double getPointX(int index){return points[index].getX();}

//устанавливает координату X точки по указанному индексу (с проверкой порядка)
1 usage
public void setPointX(int index, double x){
    if (index > 0 && x <= points[index - 1].getX()) return;
    if (index<points.length-1 && x >= points[index + 1].getX()) return;
    points[index].setX(x);
}

//возвращает координату Y точки по указанному индексу
1 usage
public double getPointY(int index){return points[index].getY();}

//устанавливает координату Y точки по указанному индексу
2 usages
public void setPointY(int index, double y){
    if (index < points.length && index >= 0) points[index].setY(y);
}
```

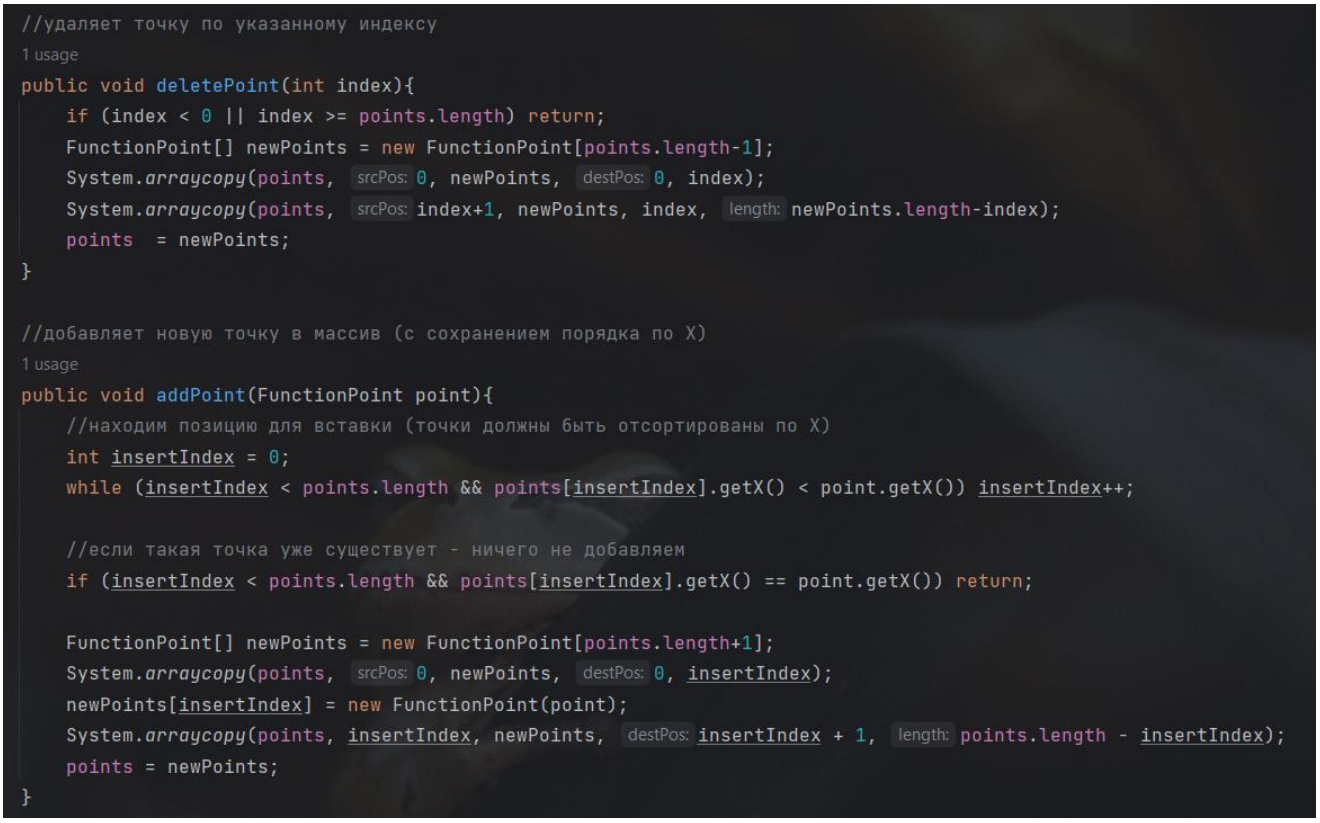
Рис. 9

Задание 6

Для выполнения этого задания добавляем в наш класс методы, изменяющие количество точек табулированной функции (Рис. 10).

Метод `deletePoint(int index)` для удаления заданной точки и метод `addPoint(FunctionPoint point)` для добавления новой точки табулированной функции. При написании метода обращаем особое внимание на обеспечение инкапсуляции, а также на сохранение упорядоченности массива точек по значениям `x`.

Для реализации этих функций я воспользовалась методом `arraycopy()` класса `System`. Согласно тому, что я прочитала о `System.arraycopy`, первым параметром является массив-источник; вторым параметром является позиция начала нового массива; третий параметр — массив назначения; четвертый параметр является начальным положением целевого массива; последний параметр — количество элементов, которые будут скопированы.



```java
//удаляет точку по указанному индексу
1 usage
public void deletePoint(int index){
    if (index < 0 || index >= points.length) return;
    FunctionPoint[] newPoints = new FunctionPoint[points.length-1];
    System.arraycopy(points, srcPos: 0, newPoints, destPos: 0, index);
    System.arraycopy(points, srcPos: index+1, newPoints, index, length: newPoints.length-index);
    points  = newPoints;
}

//добавляет новую точку в массив (с сохранением порядка по X)
1 usage
public void addPoint(FunctionPoint point){
    //находим позицию для вставки (точки должны быть отсортированы по X)
    int insertIndex = 0;
    while (insertIndex < points.length && points[insertIndex].getX() < point.getX()) insertIndex++;

    //если такая точка уже существует - ничего не добавляем
    if (insertIndex < points.length && points[insertIndex].getX() == point.getX()) return;

    FunctionPoint[] newPoints = new FunctionPoint[points.length+1];
    System.arraycopy(points, srcPos: 0, newPoints, destPos: 0, insertIndex);
    newPoints[insertIndex] = new FunctionPoint(point);
    System.arraycopy(points, insertIndex, newPoints, destPos: insertIndex + 1, length: points.length - insertIndex);
    points = newPoints;
}
```

Рис. 10

Задание 7

В последнем задании данной лабораторной требуется проверить работу класса табулированной функции. Для этого также стоит прописать класс `Main`, который будет содержать точку входа программы (Рис. 11).

В методе `main()` создаём экземпляр класса и заполняем его значениями известной табулированной функции. В моём случае это $y = x^2$. Область определения у своей функции я задаю как `[-6.0, 6.0]`, а точек в ней будет `9`.

Для удобства вывода значений я использовала отдельный метод `printFunction(TabulatedFunction function)`, который последовательно выводит значения всех точек переданной в него функции (Рис. 12).

Проверяем последовательно все методы и выводим нужные значения в консоль (Рис. 13, 14).

```
import functions.TabulatedFunction;
import functions.FunctionPoint;

public class Main {
    public static void main(String[] args) {
        //создаём экземпляр класса табулированной функции и заполняем его значениями для y=x^2
        TabulatedFunction function = new TabulatedFunction( leftX: -6.0,  rightX: 6.0,  pointsCount: 9);

        for (int i = 0; i < function.getPointsCount(); i++){
            double x = function.getPointX(i);
            function.setPointY(i,  y: function.getPointX(i)*function.getPointX(i));
        }
```

Рис. 11

```java
public static void printFunction(TabulatedFunction function){
    for (int i = 0; i < function.getPointsCount(); i++){
        System.out.println((i+1)+" точка: ("+function.getPointX(i)+", "+function.getPointY(i)+")");
    }
}
```

Рис. 12

```java
//демонстрация изначально заданной функции
System.out.println("Функция y=x^2 из "+function.getPointsCount()+" точек на отрезке [-6.0, 6.0]: ");
printFunction(function);
System.out.println();

//демонстрация значений функции в конкретных точках
double[] testValues = {2.0, -10.0, 5.7};
for (double testValue : testValues) {
    System.out.println("Значение функции в точке f(" + testValue + "): " + function.getFunctionValue(testValue));
}
System.out.println();

//демонстрация границ области определения функции
System.out.println("Функция определена на отрезке ["+function.getLeftDomainBorder()+", "+function.getRightDomainBorder()+"]");
System.out.println();

//демонстрация функции после удаления точки
function.deletePoint( index: 7);
System.out.println("Функция после удаления 8ой точки: ");
printFunction(function);
System.out.println();

//демонстрация функции после вставки точки
FunctionPoint testPoint1 = new FunctionPoint( x: -5.8, y: 21);
function.addPoint(testPoint1);
System.out.println("Функция после вставки точки с координатами (-5.8, 18.33):");
printFunction(function);
System.out.println();
```

Рис. 13

```java
//демонстрация функции после замены точки
FunctionPoint testPoint2 = new FunctionPoint( x: 2.8, y: 100);
function.setPoint( index: 6, testPoint2);
System.out.println("Функция после замены 7ой точки:");
printFunction(function);
System.out.println();

//демонстрация функции после замены конкретных координат (отдельно x и отдельно y)
function.setPointX( index: 2,  x: -4.0);
function.setPointY( index: 5,  y: 12.345);
System.out.println("Функция после замены координат X у 3 точки и Y у 6 точки: ");
printFunction(function);
System.out.println();
```

Рис. 14